UNIT – 1 INTRODUCTION TO JAVA

- Java is related to C++, which is a direct descendent of C
- From C, Java derives its syntax
- Many of Java's object-oriented features were influenced by C++
- Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991
- This language was initially called "Oak" but was renamed "Java" in 1995
- Java can be used to create two types of programs: ***applications*** and ***applets***
- An *application* is a program that runs on your computer, under the operating system of that computer
- An *applet* is an application designed to be transmitted over the Internet and executed by a Java-compatible Web browser
- An applet is actually a tiny Java program, dynamically downloaded across the network, just like an image, sound file, or video clip
- An applet is a program that can react to user input and dynamically change—not just run the same animation or sound over and over

Security:

- When you use a Java-compatible Web browser, you can safely download Java applets without fear of viral infection or malicious intent
- Java achieves this protection by confining a Java program to the Java execution access to other parts of the computer environment and not allowing it
- Java is portable across many types of computers and operating systems that are in use throughout the world

Java's Magic: The Bytecode

- The output of a Java compiler is not executable code ; rather, it is ***bytecode***
- *Bytecode* is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the ***Java Virtual Machine*** (*JVM*)

JVM is an interpreter for bytecode

- Translating a Java program into bytecode helps makes it much easier to run a program in a wide variety of environments

- The reason is straightforward: only the JVM needs to be implemented for each platform

- When a program is interpreted, it generally runs substantially slower than it would run if compiled to executable code

- The use of bytecode enables the Java run-time system to execute programs much faster than you might expect
- Sun supplies its Just In Time (JIT) compiler for bytecode, which is included in the Java 2 release

- When the JIT compiler is part of the JVM, it compiles bytecode into executable code in real time, on a piece-by-piece, demand basis

- The JIT compiles code as it is needed, during execution

The Java Buzzwords

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted
- High performance
- Distributed
- Dynamic

Simple

- If you already understand the basic concepts of object-oriented programming, learning Java will be even easier

- Because Java inherits the C/C++ syntax and many of the object-oriented features of C++, most programmers have little trouble learning Java

- Beyond its similarities with C/C++, Java has another attribute that makes it easy to learn: it makes an effort not to have *surprising* features

Object oriented

- The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance nonobjects

Robust

- Ability to create robust programs was given a high priority in the design of Java

- To better understand how Java is robust, consider two of the main reasons for program failure: ***memory management mistakes*** and ***mishandled exceptional conditions*** (that is, run-time errors)

- Memory management can be a difficult, tedious task in traditional programming environments

- For example, in C/C++, the programmer must manually allocate and free all dynamic memory

- Programmers will either forget to free memory that has been previously allocated or, worse, try to free some memory that another part of their code is still using

- Java virtually eliminates these problems by managing memory allocation and deallocation for you

- Java provides object-oriented exception handling

Multithreaded

- Java supports multithreaded programming, which allows you to write programs that do many things simultaneously

- The Java run-time system comes with an elegant yet sophisticated solution for multiprocess synchronization that enables you to construct smoothly running interactive systems

Architecture-Neutral

- Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction

- Java Virtual Machine in an attempt to alter this situation

- Their goal was “write once; run anywhere, any time, forever.”

Interpreted and High Performance

- Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode
- This code can be interpreted on any system that provides a Java Virtual Machine
- the Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using a just-in-time compiler

Distributed

- Java is designed for the distributed environment of the Internet, because it handles TCP/IP protocols
- ***Remote Method Invocation (RMI)*** feature of Java brings an unparalleled level of abstraction to client/server programming

Dynamic

- Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time

An Overview of Java

- Object-oriented programming is at the core of Java
- all computer programs consist of two elements: ***code*** and ***data***
- A program can be conceptually organized around its code or around its data
- That is, some programs are written around “what is happening” and others are written around “who is being affected.”
- The first way is called the ***process-oriented model***
- The process-oriented model can be thought of as *code acting on data*
- The second approach, ***Object-oriented programming*** organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data

The Three OOP Principles:

- ***Encapsulation*** - is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse

- ***Inheritance*** - the process by which one object acquires the properties of another object

- ***Polymorphism*** - is a feature that allows one interface to be used for a general class of actions

A First Simple Program

```
/* This is a simple Java program.  Call this file "Example.java".*/
class Example {
// Your program begins with a call to main().
public static void main(String args[]) {
System.out.println("This is a simple Java program.");
} }
```

Compiling the Program

- ***C:\>javac Example.java***

- The javac compiler creates a file called ***Example.class*** that contains the bytecode version of the program

- The output of **javac** is not code that can be directly executed

- To actually run the program, you must use the Java interpreter, called **java**.

- ***C:\>java Example***

- When a class member is preceded by **public**, then that member may be accessed by code outside the class in which it is declared

- The keyword **static** allows **main( )** to be called without having to instantiate a particular instance of the class

- The keyword **void** simply tells the compiler that **main( )** does not return a value

A Second Short Program

```
class Example2 {
public static void main(String args[]) {
int num; // this declares a variable called num
```

```
num = 100; // this assigns num the value 100
System.out.println("This is num: " + num);
num = num * 2;
System.out.print("The value of num * 2 is ");
System.out.println(num);
} }
```

Using Blocks of Code:

Using { and }

Lexical Issues

- ***Whitespace:***
  - Java is a free-form language
  - In Java, whitespace is a space, tab, or newline

- ***Identifiers:***
  - Identifiers are used for class names, method names, and variable names
  - Java is case-sensitive

Literals:

- A constant value in Java is created by using a *literal* representation of it

Comments

- There are three types of comments defined by Java.
- Single-line and multiline
- The third type is called a *documentation comment*
- This type of comment is used to produce an HTML file that documents your program
- The documentation comment begins with a **/**** and ends with a ***/**

Separators

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

The Java Keywords

- There are 49 reserved keywords currently defined in the Java language

abstract	continue	goto	package	synchronized
assert	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	

Data Types, Variables, and Arrays

Java Is a Strongly Typed Language

- Every variable has a type, every expression has a type, and every type is strictly defined
- All assignments, whether explicit or via parameter passing in method calls, are checked for type compatibility
- The Java compiler checks all expressions and parameters to ensure that the types are compatible

The Simple Types --- Java defines eight simple (or elemental) types of data: **byte**, **short**, **int**, **long**, **char**, **float**, **double**, and **boolean**

Integers

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

```
class Light {
public static void main(String args[]) {
int lightspeed;
long days;
long seconds;
long distance;
// approximate speed of light in miles per second
lightspeed = 186000;
days = 1000; // specify number of days here
seconds = days * 24 * 60 * 60; // convert to seconds
distance = lightspeed * seconds; // compute distance
System.out.print("In " + days);
System.out.print(" days light will travel about ");
System.out.println(distance + " miles.");
} }
```

Floating-Point Types

Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

```
class Area {
public static void main(String args[ ]) {
```

```
double pi, r, a;
r = 10.8; // radius of circle
pi = 3.1416; // pi, approximately
a = pi * r * r; // compute area
System.out.println("Area of circle is " + a);
} }
```

Characters

- **char** in Java is not the same as **char** in C or C++.
- In C/C++, **char** is an integer type that is 8 bits wide
- Instead, Java uses Unicode to represent characters
- *Unicode* defines a fully international character set that can represent all of the characters found in all human languages
- In Java **char** is a 16-bit type
- The range of a **char** is 0 to 65,536
- There are no negative **char**s

```
class CharDemo {
public static void main(String args[ ]) {
char ch1, ch2;
ch1 = 88; // code for X
ch2 = 'Y';
System.out.print("ch1 and ch2: ");
System.out.println(ch1 + " " + ch2);
} }
```

```
class CharDemo2 {
public static void main(String args[ ]) {
char ch1;
ch1 = 'X';
System.out.println("ch1 contains " + ch1);
ch1++; // increment ch1
System.out.println("ch1 is now " + ch1);
} }
```

Booleans

- Can have only one of two possible values, **true** or **false**

```
class BoolTest {
public static void main(String args[]) {
boolean b;
b = false;
```

```
System.out.println("b is " + b);
b = true;
System.out.println("b is " + b);
// a boolean value can control the if statement
if(b) System.out.println("This is executed.");
b = false;
if(b) System.out.println("This is not executed.");
// outcome of a relational operator is a boolean value
System.out.println("10 > 9 is " + (10 > 9));
} }
```

The output generated by this program is shown here:

```
b is false
b is true
This is executed.
10 > 9 is true
```

- Integer Literals – Decimal, Hexa and Octal

- Floating-Point Literals
  - For example, 2.0, 3.14159, and 0.6667 represent valid standard - notation floating-point numbers

Scientific notation uses a standard-notation, floating-point number plus a suffix that specifies a power of 10 by which the number is to be multiplied eg. 6.022E23, 314159E–05, and 2e+100

- Boolean Literals – true , false

- Character Literals
  - Characters in Java are indices into the Unicode character set
  - They are 16-bit values that can be converted into integers and manipulated with the integer operators, such as the addition and subtraction operators

- A literal character is represented inside a pair of single quotes

Escape Sequence	Description
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)
\'	Single quote
\"	Double quote
\\	Backslash
\r	Carriage return
\n	New line (also known as line feed)
\f	Form feed
\t	Tab
\b	Backspace

String Literals

- "Hello World"
- "two\nlines"
- "\"This is in quotes\""

Declaring a Variable

type identifier [= *value*][, *identifier* [= *value*] ...] ;

```
int a, b, c;            // declares three ints, a, b, and c.
int d = 3, e, f = 5;    // declares three more ints, initializing
                        // d and f.
byte z = 22;            // initializes z.
double pi = 3.14159;    // declares an approximation of pi.
char x = 'x';           // the variable x has the value 'x'.
```

Dynamic Initialization

```
class DynInit {
public static void main(String args[]) {
double a = 3.0, b = 4.0;
// c is dynamically initialized
double c = Math.sqrt(a * a + b * b);
```

```
System.out.println("Hypotenuse is " + c);
} }
```

The Scope and Lifetime of Variables

- Java allows variables to be declared within any block
- A block is begun with an opening curly brace and ended by a closing curly brace
- A block defines a *scope*
- In Java, the two major scopes are those defined by a class and those defined by a method
- As a general rule, variables declared inside a scope are not visible (that is, accessible) to code that is defined outside that scope
- Scopes can be nested
- Objects declared in the outer scope will be visible to code within the inner scope ; but reverse not true

```
class Scope {
public static void main (String args[]) {
int x; // known to all code within main
x = 10;
if(x == 10) { // start new scope
int y = 20; // known only to this block
// x and y both known here.
System.out.println("x and y: " + x + " " + y);
x = y * 2;
}
// y = 100; // Error! y not known here
// x is still known here.
System.out.println("x is " + x);  } }
class LifeTime {
public static void main(String args[]) {
int x;
for(x = 0; x < 3; x++) {
int y = -1; // y is initialized each time block is entered
System.out.println("y is: " + y); // this always prints -1
y = 100;
System.out.println("y is now: " + y);
} } }


class ScopeErr {
public static void main(String args[]) {
int bar = 1;
{ // creates a new scope
int bar = 2; // Compile-time error – bar already defined!
} } }
```

Type Conversion and Casting

- When one type of data is assigned to another type of variable, an *automatic type conversion* will take place if the following two conditions are met:
    - The two types are compatible.
    - The destination type is larger than the source type.

Casting Incompatible Types

- What if you want to assign an **int** value to a **byte** variable?

- This conversion will not be performed automatically, because a **byte** is smaller than an **int**

(*target-type*) *value*

- *target-type* specifies the desired type to convert the specified value to

```
int a;
byte b;
// ...
b = (byte) a;
```

```
class Conversion {
public static void main(String args[]) {
byte b;
int i = 257;
double d = 323.142;
System.out.println("\nConversion of int to byte.");
b = (byte) i;
System.out.println("i and b " + i + " " + b);
System.out.println("\nConversion of double to int.");
i = (int) d;
System.out.println("d and i " + d + " " + i);
System.out.println("\nConversion of double to byte.");
b = (byte) d;
System.out.println("d and b " + d + " " + b);
}
}
```

Output:

```
Conversion of int to byte.
i and b 257 1
Conversion of double to int.
d and i 323.142 323
```

Conversion of double to byte.
d and b 323.142 67

Automatic Type Promotion in Expressions

Example:
byte a = 40;
byte b = 50;
byte c = 100;
int d = a * b / c;

- Java automatically promotes each **byte** or **short** operand to **int** when evaluating an expression
- Subexpression **a * b** is performed using integer

byte b = 50
b = b * 2
// error: Cannot assign an int to a byte

In this case we need to explicitly specify:

```
byte b = 50;
b = (byte) (b*2);
```

The Type Promotion Rules

- All **byte** and **short** values are promoted to **int**
- If one operand is a **long**, the whole expression is promoted to **long**
- If one operand is a **float,** the entire expression is promoted to **float**
- If any of the operands is **double**, the result is **double**

```
class Promote {
public static void main(String args[ ]) {
byte b = 42;
char c = 'a';
short s = 1024;
int i = 50000;
float f = 5.67f;
double d = .1234;
double result = (f * b) + (i / c) - (d * s);
System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
System.out.println("result = " + result);
}
}
```

double result = (f * b) + (i / c) - (d * s);

- In the first subexpression, **f * b**, **b** is promoted to a **float** and the result of the subexpression is **float**
- Next, in the subexpression **i / c**, **c** is promoted to **int**, and the result is of type **int**
- In **d * s**, the value of **s** is promoted to **double**, and the type of the subexpression is **double**
- The outcome of **float** plus an **int** is a **float**
- Then the resultant **float** minus the last **double** is promoted to **double**, which is the type for the final result of the expression